ProToS: Automation of Flight Control Procedures for the European Data Relay System

Philipp Hamacher[*] Thorsten Beck[†]

Deutsches Zentrum für Luft- und Raumfahrt e. V., German Aerospace Center
Münchener Straße 20, 82234 Weßling, Germany

The European Data Relay System (EDRS) offers a high-speed communication service between satellites, spacecraft, UAVs, and ground stations, using a relay satellite constellation in geostationary orbit, equipped with high-end laser communication terminals (LCTs). As part of the ESA ARTES 7 program, EDRS is realized as a public private partnership with DLR GSOC responsible for the Devolved Payload Control Center (DPCC) of EDRS-A, and the Satellite Control Center (SCC) of EDRS-C, the latter being scheduled for launch in 2018.

The EDRS service level agreement foresees continuous payload utilization, up to 200 links per day over the expected lifetime of 15 years, with a targeted service availability of at least 99.6% over 60 days and an order reaction time well below one hour; a task that can hardly be managed in a manual or semi-automated operations concept. All payload and most platform operations for EDRS-C are therefore executed using a procedure automation framework, the GSOC Procedure Tool Suite (ProToS).

This paper shall give a conceptional overview of the ProToS execution and automation framework as well as the mission specific configuration for EDRS-C.

I. Introduction

The automation of spacecraft monitoring and control operations has become increasingly important at the German Space Operations Center (GSOC). Currently, two different approaches are being used, depending on the satellite's orbit. An example for the automation of a low earth orbit (LEO) mission is TerraSAR-X/TanDEM-X which extensively makes use of automatic commanding. Among various ground support activities, the main task for this LEO automation is to establish the command link setup and to reliably send the prepared timeline of telecommands (TCs) to the on-board schedule.[3] A different automation concept is used for missions in geostationary orbit (GEO); since there is a continuous TM/TC link available, routinely performed tasks can be triggered and executed automatically in real-time which allows for additional functionality, including analysis and reaction to live telemetry. This was already implemented at GSOC for the mission EDRS-A.[9] This GEO automation concept is currently the prime focus for ProToS because it is expected to offer the most potential in saving work effort for routinely performed tasks.

The monitoring and control system (MCS) "GECCOS",[2] which is operational at GSOC, was contiuously modernized and enhanced since it was forked from ESA's SCOS-2000. This system already provides a mechanism to support simple command execution flows based on TM parameters and TC acknowledges; nevertheless, this mechanism was only partially used in an operational surrounding because generation of compatible TC sequences is complex and the monitoring during execution is not intuitive. This problem was approached by using an external tool which allows the definition of complex execution workflows in form of flight control procedures (FCPs). Initially, the commercial software MOIS[7] was used for this purpose. The FCPs would then be exported to an MCS compatible format and processed by multiple tools until they could be executed.

[*]Mission Control and Data Systems Engineer, Mission Operations Department, German Space Operations Center
[†]System Engineer, ARPnet IT Services UG, 82234 Weßling

On the basis of the MOIS FCP exchange format, ProToS was developed at GSOC[1] to fulfill required constraints which arose for checkout activities for the mission Eu:CROPIS. Today, ProToS allows user friendly access to all necessary aspects of FCP activities which includes generation, instantiation, execution and automation in a single tool. This integrated architecture reduces error sources by minimizing the number of software interfaces and facilitates the workflow for the operation engineers. Additionally to this integrated FCP management, special focus was given to increase functionality and flexibility of the automation engine to allow modification of operation tasks during system runtime. In the future, the system is expected to reliably manage the complex task of commanding and supervising the continuous payload operations for EDRS-C.

II. System Architecture

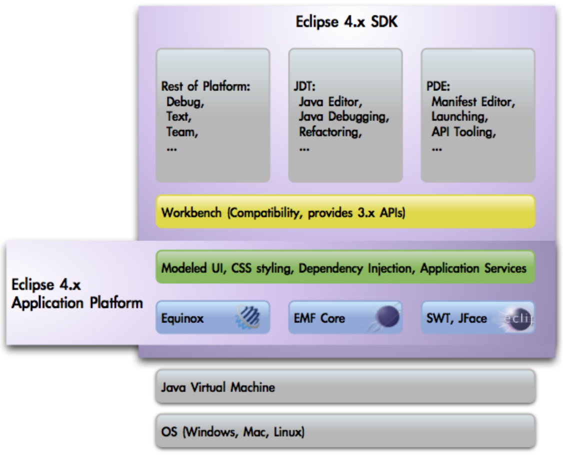
This section shall provide an overview of the main ProToS software components as well as the operational surrounding under which the software will be deployed.

A. Technologies

ProToS is written in Java and is based on the Eclipse Rich Client Platform (RCP). Eclipse, widely known as an integrated development environment (IDE), was built in a modular way, allowing reusage of its abundant and well tested core components. The minimum set of these core components resembles the RCP platform.[10]

For ProToS, the revised Eclipse4 (e4) version is being used which offers additional functionality like a dependency injection framework, a dynamic graphical user interface (GUI) or CSS styling. An overview of the e4 technologies can be seen in figure 1.

A large variety of open-source plugins is available as extensions for the e4 platform under the commercial-friendly Eclipse Public License[12] (EPL). The following plugins are used within ProToS:



Eclipse 4.x SDK
Rest of Platform: Debug, Text, Team, ...
JDT: Java Editor, Java Debugging, Refactoring, ...
PDE: Manifest Editor, Launching, API Tooling, ...
Workbench (Compatibility, provides 3.x APIs)
Eclipse 4.x Application Platform
Modeled UI, CSS styling, Dependency Injection, Application Services
Equinox
EMF Core
SWT, JFace
Java Virtual Machine
OS (Windows, Mac, Linux)

Figure 1: Eclipse4 SDK[11]

- CDO:[4] An extension for the eclipse modeling framework (EMF). This plug-in allows to share data objects among multiple clients. All changes are handled by a transaction based modification mechanism which is persisted by either an object oriented or relational database backend. A locking framework allows safe modification of objects in a multi-client environment. Object repositories of arbitrary size are supported by using a caching mechanism to only load required objects into the computer's memory.
- Nebula Nattable:[5] A highly configurable spreadsheet viewer and editor.
- Nebula Grid:[6] A GUI widget which is used in ProToS to display FCPs in the editor as well as in the executor. The tree-like structure allows to display the logical control flow of a procedure and a custom renderer can be used to show graphical elements which are not part of the standard implementation.

B. ProToS Software Architecture

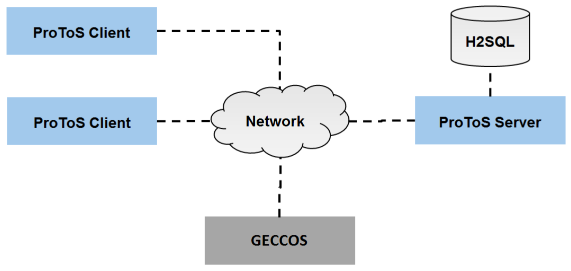
ProToS shall support all main activities related to flight control procedures which consist at GSOC of the following four tasks:

1. Procedure generation and version control
2. Instantiation of variable values

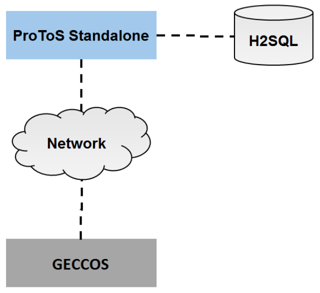
3. Procedure execution
4. Automation activities

Each of these tasks requires its own graphical user interface. For this, ProToS makes use of the Eclipse4 dynamic GUI model which allows the definition of individual perspectives. The idea of such perspectives is, that the same window can be used to display different elements, depending on the task that is to be performed. The user can easily switch from one perspective to the other without losing the respective state of each task.

ProToS is realized as a client/server application. The default use-case of multiple clients connecting to a common server is shown in figure 2a. This architecture allows many subsystem engineers to collaboratively work on FCPs without adjacent change merging and validation strategies. For some setups, it may be desirable to use ProToS in a single-user environment in order to keep the system as simple as possible (figure 2b). The modular Eclipse plug-in framework allows ProToS to include server and client in a single application, effectively making it a standalone application. This is useful for subsystem engineers who may need to develop procedures without currently being able to connect to the network. Another use-case was the setup for assembly integration and test (AIT) facilities for the mission Eu:CROPIS.



(a) Client/Server



(b) Standalone

Figure 2: Deployment scenarios[1]

An overview of the main interactions between the three different software instances: client, server and MCS can be seen in figure 3. The functionality of CDO (section II A) to share data objects among multiple clients is essential for the ProToS application workflow and is used for the communication between server and client. The interface to the MCS is realized by using the middleware CORBA (Common Object Request Broker Architecture).

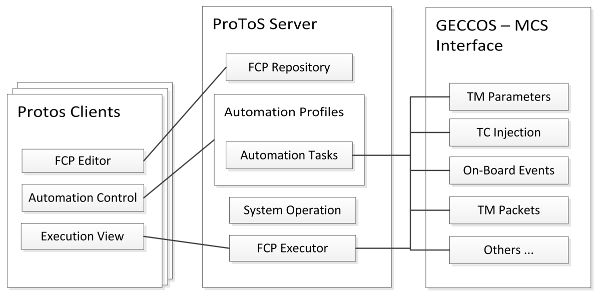


Figure 3: Operational overview

By the approach to perform the client-server communication via state changes on the shared data model, it is possible to ensure that all clients are seeing the same information without the implementation of complex notification based state machines. For critical changes, a locking mechanism enforces that only one client may modify the content of an object. For non-critical changes, CDO's fail-safe transaction based modification method is sufficient which will only allow changes without any intermediary commits from other sources. If any content of a transaction cannot be committed, the responsible source is notified with an appropriate exception. The client can then decide to roll-back any local changes or to resolve the conflicts by a merge strategy. A good example is the current implementation of the FCP executor. Once a client decides to start the execution of an FCP, it will set a flag in the appropriate shared object and commit the changes to CDO. This change will be received by the ProToS server as well as all other clients. The server will check the request and then start or deny the execution of the procedure and finally set another status flag accordingly.

This process is shown in figure 4 where the "Execution State" and "Procedure Object" elements represent the previously mentioned shared EMF objects. All changes in the execution state will be available to any client connected to the server, making it able to monitor the whole process. The currently executed procedure is available to all clients who open the execution perspective.

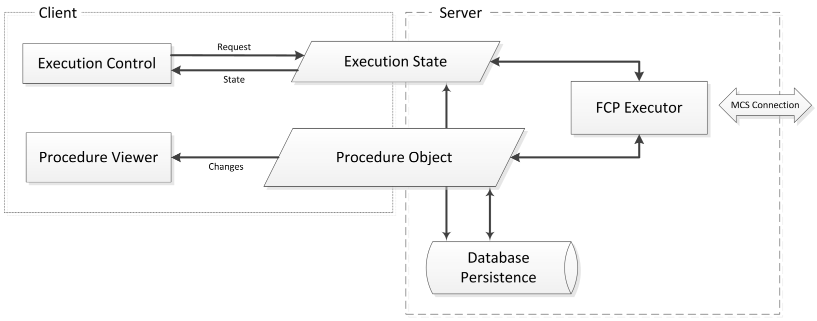
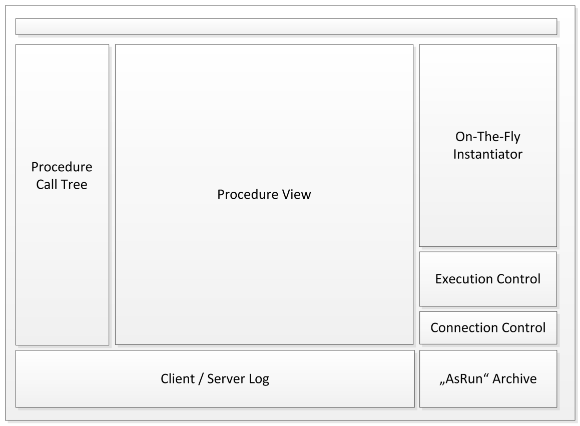


Figure 4: Server/Client communication via shared data containers

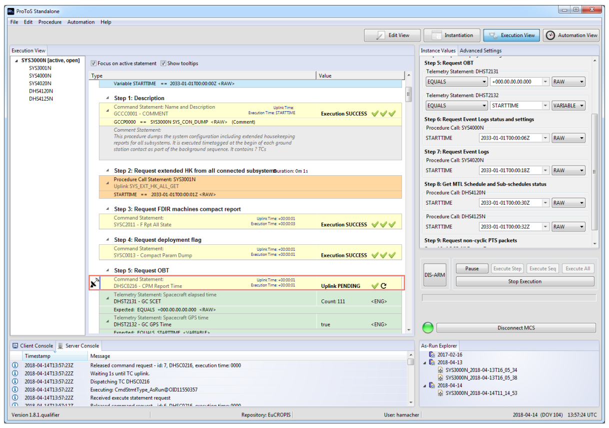
The schematic design of the execution perspective is shown in figure 5a and an actual example is shown in figure 5b. For clients that only observe the execution process, the "Procedure Call Tree" and the "Procedure View" parts are the only relevant parts. The "Procedure View" displays the "Procedure Object" (figure 4) which is currently in-line for execution. If the procedure contains references to other procedures (i.e. procedure calls), a tree structure containing all sub-procedures is shown within the "Procedure Call Tree" control. A client may inspect any of the loaded procedures without affecting the execution process.

The actual FCP execution logic is running directly on the ProToS server. This ensures, that no active client is required in order to perform automated tasks and it minimizes the communication delay to the MCS. During FCP execution, a configurable state flow calculator decides which statement is to be run next, depending on the outcome of the previous statements, much like it is the case in a programming language. This includes processing of step types like "if-then-else" or "while" routines. Configurable, for example, is the behavior after failed TCs or TM checks. The execution may be required to continue or it may be necessary to do a safety stop, depending on the current operational scenario. Any of the authorized clients may then decide to resume or to cancel the FCP execution via the "Execution Control" (figure 5a) which is able to change the "Execution State" shown in figure 4.

The "On-The-Fly Instantiator" control can be used to modify variable content of the procedure like TM values or TC parameters, and it is mainly used for AIT activities. During routine operations, it will not be part of the perspective because instantiation is considered to be a designated operational step before an FCP is approved for execution. The AsRun archive on the bottom right allows quick access to historic data which can be loaded to the regular procedure view for inspection and re-execution.



(a) Schematic Design



SYS3000N [active, open]
SYS3001N
SYS4000N
SYS4020N
DHS4120N
DHS4125N

Focus on active statement ✓ Show tooltips ✓

Type	Value
Variable STARTTIME == 2033-01-01T00:00:00Z <RAW>	
Step 1: Description	
Command Statement: Name and Description GCCC0001 - COMMENT — Uplink Time: Execution Time: STARTTIME	Execution SUCCESS ✓✓✓
GCCP0000 == SYS3000N SYS_CON_DUMP <RAW> (Comment)	
Comment Statement: *This procedure dumps the system configuration including extended housekeeping reports for all subsystems. It is executed timetagged at the begin of each ground station contact as part of the background sequence. It contains ? TCs*	
Step 2: Request extended HK from all connected subsystems Duration: 0m 1s	
Procedure Call Statement: SYS3001N Uplink SYS_EXT_HK_ALL_GET	
STARTTIME == 2033-01-01T00:00:01Z <RAW>	
Step 3: Request FDIR machines compact report	
Command Statement: SYSC2011 - F Rpt All State — Uplink Time: +00:00:01 Execution Time: +00:00:03	Execution SUCCESS ✓✓✓
Step 4: Request deployment flag	
Command Statement: SYSC0013 - Compact Param Dump — Uplink Time: +00:00:01 Execution Time: +00:00:01	Execution SUCCESS ✓✓✓
Step 5: Request OBT	
Command Statement: DHSC0216 - CPM Report Time — Uplink Time: +00:00:01 Execution Time: +00:00:01	Uplink PENDING ✓
Telemetry Statement: Spacecraft elapsed time DHST2131 - GC SCET	Count: 111 <ENG>
Expected: EQUALS +000.00.00.00.000 <RAW>	
Telemetry Statement: Spacecraft GPS time DHST2132 - GC GPS Time	true <ENG>
Expected: EQUALS STARTTIME <VARIABLE>	

Step 5: Request OBT
Telemetry Statement: DHST2131 — EQUALS | +000.00.00.00.000 | RAW
Telemetry Statement: DHST2132 — EQUALS | STARTTIME | VARIABLE
Step 6: Request Event Logs status and settings
Procedure Call: SYS4000N — STARTTIME | 2033-01-01T00:00:06Z | RAW
Step 7: Request Event Logs
Procedure Call: SYS4020N — STARTTIME | 2033-01-01T00:00:18Z | RAW
Step 8: Get MTL Schedule and Sub-schedules status
Procedure Call: DHS4120N — STARTTIME | 2033-01-01T00:00:30Z | RAW
Procedure Call: DHS4125N — STARTTIME | 2033-01-01T00:00:32Z | RAW
Step 9: Request non-cyclic PTS packets

Timestamp	Message
2018-04-14T13:57:23Z	Released command request - id: 7, DHSC0216, execution time: 0000
2018-04-14T13:57:22Z	Waiting 1s until TC uplink.
2018-04-14T13:57:22Z	Dispatching TC DHSC0216
2018-04-14T13:57:22Z	Executing: CmdStmtType_AsRun@OID11550357
2018-04-14T13:57:22Z	Received execute statement request

2017-02-16
2018-04-13
SYS3000N_2018-04-13T16_05_34
SYS3000N_2018-04-13T16_05_38
2018-04-14
SYS3000N_2018-04-14T11_14_53

Version 1.8.1.qualifier | Repository: EuCROPIS | User: hamacher | 2018-04-14 (DOY 104) 13:57:24 UTC

(b) Implementation

Figure 5: ProToS Execution Perspective

C. Automation Concept

In order to fulfill the challenge to automatically manage many different tasks, including payload operations as well as some routine satellite bus operations, a highly flexible automation engine needed to be developed. A number of requirements were identified as necessary to support this goal:

- Comprehensive overview of running automation tasks
- Grouping of different tasks for different operational scenarios
- Access to live TM/TC data from the MCS
- React to and perform operating system operations (e.g. reaction to new files)
- Modification and creation of new tasks during application runtime
- Versioning as well as import/export of tasks

The ProToS client GUI provides a perspective to maintain and observe the state of any running automation task. By defining different ProToS login authorization roles, we can allow some users to see the current state of the automation, while others may be able to start/stop, change exisiting or create new tasks during runtime.

Automation tasks are grouped as logical units into different automation profiles. This architecture allows a well-defined change of the automation behavior from one mode to the other, an example being nominal and contingency mode.

A version control mechanism was implemented on top of the profile grouping. This is necessary to perform well-defined transitions from an active automation definition to a newly adjusted one, always having the possibility to revert any changes if the operational behavior is not as desired. Each version of the automation profiles can be exported and imported to a standard XML file which allows to develop automation profiles outside of the operational environment and to transfer them once they are validated and ready to be deployed.

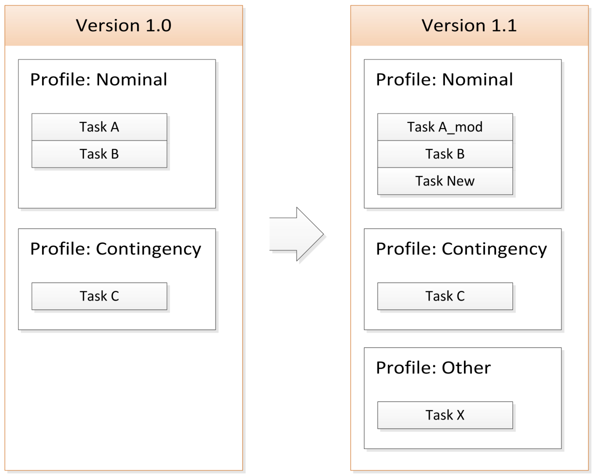


Figure 6: Automation Profiles

As a wrapper for the automated task, we need a well-defined trigger mechanism to initiate their execution. Currently, the following events are foreseen to be relevant and are implemented as available templates.

- Single execution at a given time reference
- Recurring with a given time period
- MCS event
- Flight-Procedure event
- File system event

In order to achieve the required flexibility for the actual content of the automation tasks, Java source code is being used directly as the input, meaning basically no restriction on the potential functionality. Compilation during runtime is possible by using the javax compiler API which is part of the java development kit (JDK) since Java SE 6.[8] The only restriction being, that a JDK needs to be installed on the operational ProToS server machine in contrast to the usual java runtime environment (JRE). Authorized clients will be able to attach

Java code to an automation task, and the server will perform the compilation. In case of compilation errors, the client is informed with an appropriate message. Once the code was successfully compiled, the .class file will be saved as binary content within the automation task. This process is shown in figure 7. Once the compiled class is available in the automation task, it can be run by using the Java dynamic class loading mechanism.

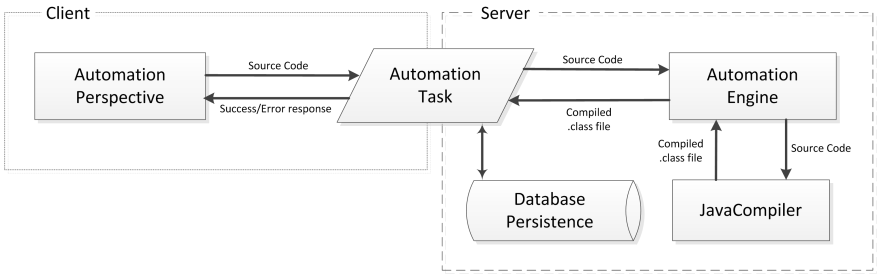


Figure 7: Server/Client communication via shared data containers

The automation interface which allows dynamic code contribution leaves open questions regarding security. Since the new code may not necessarily come from trusted sources, countermeasures against malicious content need to be taken. As a first consequence, only a few designated administration users will be able to modify source content for automation tasks. In the future, if the automation engine shall be available for external partners, a combination of secure connections via VPN, ProToS login credentials and the usage of the Java SecurityManager[8] is imaginable.

III. Automation Concept for EDRS-C

EDRS-C represents the second node of the EDRS space data-highway and is scheduled for launch in the first half of 2019. The overall coordination of nodes, the link planning and all customer interaction is carried out by the EDRS Mission Operations Center (MOC), operated by Airbus DS, thereby providing the basis for scheduling and commanding of inter-satellite link requests through the EDRS-C satellite control center (SCC).

The Satellite Control Center of EDRS-C is part of the DLR GSOC multi-mission control center environment, with DLR being responsible for the safe operation of both satellite platform and payload. As mentioned before, the EDRS service-level agreement poses very ambitious requirements on service availability, reaction times and reporting, which is why all payload and most platform operations performed by the SCC were designed to be automated, using the ProToS automation engine.

Two operational phases need to be distinguished in the EDRS-C automation concept:

1. Launch and Early Orbit Phase (LEOP): during LEOP, ProToS will aid the operations personnel in the semi-automated execution of FCPs, enabling the user to trigger and interact with the procedure control flow in real-time, receiving procedure-based telemetry feedback and command acknowledgments, all in a single view.

2. Routine operations: during routine operations, the EDRS-C satellite will be under permanent control by the ProToS automation engine. This includes all payload operations and most platform routine maintenance tasks, e.g., the execution of station keeping maneuvers (SKMs). The automation engine is designed to supervise the complete cycle of telecommand uplink and execution, as well as reaction monitoring of telemetry. Updates to the onboard mission timeline are scheduled and uplinked autonomously, typically triggered via high-level request files from ground or by on-board events.

We will focus in this section on the routine operations phase for which the following recurring and event-driven activities have been implemented in ProToS:

1. Procedure request processor (file system event-triggered)

2. Processing delay & ranging activities (recurring task)

3. Confirmation message generator (procedure event-triggered)

4. Link session info/report generator (procedure event-triggered)
5. On-board schedule report generator (file system event-triggered)
6. OBT sync request generator (MCS event-triggered)
7. LCT alignment matrix calibration (procedure event-triggered)
8. Status report generator (recurring task)

In the following, these automated activities are described in more detail.

A. Procedure request processor

The procedure request processor is triggered on occurrence of new XML request files, originating from the Mission Operations Center or the SCC-internal planning component. Each valid request file can trigger the immediate instantiation and execution of one or several flight procedures, using the requested procedure templates and the instance values provided in the request. The instantiated procedure is added to the internal execution stack, together with information on the type of request. The following EDRS request types are supported:

- Payload Link Configuration Request: MOC request, triggering the configuration and execution of link requests.
- Payload Routine Configuration Request: MOC request, triggering the generic execution of flight procedures, taken from a pre-defined subset of all procedures)
- Payload Configuration Deletion Request: MOC request, triggering the removal of requests from the on-board time line.
- Forward Tasking Data and Deletion Requests: MOC request, triggering the upload and deletion of user data, used for link requests in forward direction
- SCC Internal Request: e.g. station-keeping maneuvers, on-board time synchronization, upload of on-orbit propagtors)

The request processor represents the heart of the EDRS-C automation, since most of the satellite command load will be generated by this task. It is also the most critical task, as for instance an incorrect execution of a station-keeping maneuver can directly affect satellite health and payload safety. Therefore, any unhandled anomaly during procedure execution immediately interrupts the request execution and an operator alarm message is generated. And therefore, only a subset of EDRS-C procedures have been cleared for automated execution and this generally comes with tighter requirements regarding procedure development (safety checks, abort conditions, etc.) and procedure validation.

B. Processing delay & ranging activities

The EDRS-C ranging concept foresees a five minutes long ranging measurement executed every three hours. During ranging, it may become necessary to pause all commanding activities. Therefore a procedure execution delay will be implemented as an optional recurring task. Its execution time is synchronized with the scheduled ranging activities, i.e. every three hours starting at midnight (0:00 UTC) and with a duration of 10 minutes. During this time the task ensures that the processing of procedures is delayed which includes execution of automated procedures, as well as manually executed procedures. For procedures not already in progress, the delay affects the complete processing including telemetry checks. If a procedure is already in progress, the execution is halted only when the active statement is a telecommand. Other statements (telemetry checks, waiting steps) are executed as usual. After the task duration elapsed, processing of requests is resumed in the order of arrival.

C. Confirmation message generator

Updates to the execution state of active procedures trigger the confirmation message generator task, which is responsible for the creation of XML confirmation messages to MOC. Messages are generated per request, which may consist of more than one procedure. The request execution state is derived from all procedures in the execution history that share the same request id (i.e. of all procedures associated with a particular Payload Configuration, Forward Tasking Data, Deletion or Internal Request). This allows autonomous, quasi real-time reporting of uplink and execution states of high-level MOC requests.

D. Link session info and link session report generator

During each EDRS link session, the automation engine collects telemetry and listens to on-board event packets to generate the link session info and report messages for MOC. TM acquisition is synchronized with the execution time of link sessions and the task is therefore configured as a procedure-triggered automation task. The link session info message uses on-board events to determine the exact moment in time when the link acquisition succeeded, sending this information back to MOC in form of an XML message. Over the course of link session execution, during pointing, acquisition and communication phase, a number of session specific TM parameters are recorded. After session finalization, the recorded values are compiled into the Link Session XML Report and delivered to MOC.

E. On-board schedule report generator

In order for the SCC to schedule or delete payload configurations, the status of the on-board time-tagged command schedule has to be made available to the SCC scheduling component, i.e. the link management system (LMS).[13] Most input to this report stems directly from the on-board queue model (OBQM), modelled inside the MCS where the status of every released TTC is tracked using acknowledgments received in telemetry. The MCS writes the content of the OBQM to a report file, which is then read by this automation task. The task supplements the report with request related information and generates a new report in XML format which is forwarded to the LMS, containing for every TC its request id, TC id, APID, source sequence counter, UTC execution time and load status.

F. OBT synchronization request generator

The EDRS-C platform hosts an on-board clock which is automatically synchronizing the clock of the LCT payload. After initial synchronization however, the clock is expected to drift, so onboard time (OBT) synchronization has to be repeated each time the delta between OBT and GPS time exceeds a configurable threshold. The OBT sync request generator monitors the OBT-ground time difference, which is provided by the MCS in form of a derived parameter. As soon as the time difference exceeds a configurable threshold for a configurable number of consecutive telemetry samples, the task generates a sync request containing the time difference. This request is provided in XML format to the LMS. After request generation, a configurable dead-time is maintained in which the task does not poll the time difference, in order to account for commanding delays caused e.g. by SKMs during which time synchronization must not be scheduled. An additional safety is put in place in order to deal with large clock drifts attributed to on-board anomalies: time differences larger than n seconds (where n is configurable with default 2) do not trigger a sync request. Instead an alarm message is sent to an operator.

G. LCT alignment matrix calibration

The LCT array is exposed to thermal fluctuations and its alignment matrix thus has to be recalibrated from time to time. An alignment matrix update is calculated by the flight-dynamics system (FDS) of the SCC, based on the input provided by this automation task. During each link session the Automator will automatically collect this telemetry with a polling time of once every 4 seconds and provide a report to FDS in form of an XML file, provided that the link request was reported successful.

H. Status report generator

This automation task regularly provides the status of ground-segment, satellite platform and LCT payload to internal and external monitoring components. Telemetry is acquired with a configurable frequency (once every two minutes by default) and sent out in form of an XML file. In case of processing errors or anomalies, this task automatically generates an alarm message that notifies an operator for further investigation.

IV. Conclusion

This paper aimed to provide insight into the automation framework which will be used to handle the challenging operational requirements of the project EDRS-C. An overview of the planned automated activities was given which will be carried out during the different phases of the mission. The underlying software application ProToS was described in some technical detail, focussing on the executor and automation engine.

Reliable data on the performance of the automation framework is expected to be collected after the launch of the EDRS-C satellite, currently planned for 2019. Future work will include usability optimizations of ProToS in order to create a platform which is suited to support different needs in a multi-mission environment. Additionally, it shall be investigated if and how it may be possible to provide encapsulated ProToS functionality to external partners outside of the control room. Different scenarios are imaginable and need to be thoroughly examined especially regarding security constraints.

Glossary

TM Telemetry

TC Telecommand

LEO Low Earth Orbit

GEO Geostationary Orbit

GSOC German Space Operation Center

EDRS European Data Relay Satellite System

ProToS Procedure Tool Suite

FCP Flight Control Procedure

MCS Monitoring and Control System

GECCOS GSOC Enhanced Command and Control System for Operating Spacecrafts

ESA European Space Agency

EMF Eclipse Modeling Framework

CDO Connected Data Objects

CSS Cascading Style Sheets

AIT Assembly, Integration and Test

MOC Mission Operation System

SCC Satellite Control Center

LMS Link Management System

FDS Flight Dynamics System

OBT On-board time

LCT Laser Communication Terminal

Acknowledgements

The authors would like to acknowledge the other two ProToS team members Leonard Schlag and Stefan Gärtner, who made substantial contributions to the success of this software project.

References

[1]Beck T., Schlag L., Hamacher, P. "ProToS: Next Generation Procedure Tool Suite for Creation, Execution and Automation of Flight Control Procedures," *SpaceOps 2016, Daejon, Korea, 2016*

[2]Stangl C., Lotko B., Geyer M., Braun A., Oswald M. "GECCOS – the new Monitoring and Control System at DLR-GSOC for Space Operations, based on SCOS-2000," *SpaceOps 2014, Pasadena, U.S.A., 2014*

[3]Zimmermann S., Schulze D., Stangl C., "Command Chain Automation," *SpaceOps 2014, Pasadena, U.S.A., 2014*

[4]CDO Model Repository, *https : //www.eclipse.org/cdo/*

[5]Nebula Nattable, *https : //www.eclipse.org/nattable/*

[6]Nebula Grid, *https : //www.eclipse.org/nebula/widgets/grid/grid.php*

[7]RHEA Group, "White Paper: MOIS - Manufacturing and Operations Information System," *2014,* $http://www.rheagroup.com/wp-content/uploads/2015/07/MOIS_White_paper.pdf$

[8]Java SE 1.8 Documentation, *https : //docs.oracle.com/javase/8/docs/api/*

[9]Beck, T., Schmidhuber, M., Scharringhausen, J.C., "Automation of Complex Operational Scenarios - Providing 24/7 Inter-Satellite Links with EDRS," *SpaceOps 2016 Conference, Daejeon, Korea, 2016*

[10]Eclipse Foundation Wiki, "Rich Client Platform," *https : //wiki.eclipse.org/Rich_Client_Platform*

[11]Eclipse Foundation Wiki, "Eclipse 4 SDK," *https : //wiki.eclipse.org/Eclipse4*

[12]Eclipse Public License - v 1.0, *https : //eclipse.org/org/documents/epl − v10.php*

[13]Göttfert, T., Wörle, M. T., Prüfer, S., Lenzen, S., "Operating and Evolving the EDRS Payload and Link Management System," *SpaceOps 2018 Conference, Marseille, France, 2018*